

Calcolatori Elettronici T Ingegneria Informatica

01 Introduzione



Docenti

Stefano Mattoccia

Ricevimento: via Teams, su appuntamento concordato via email

Email: stefano.mattoccia@unibo.it

Home page: <http://vision.disi.unibo.it/~smatt>

Andrea Conti (tutor, AA 2021/22)

Ricevimento: via Teams, su appuntamento concordato via email

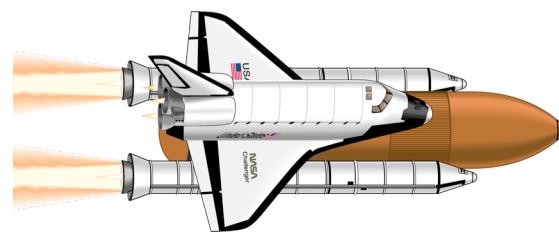
Email: andrea.conti35@unibo.it

Obiettivi del corso

Apprendere,

- i principi di funzionamento
- le architetture
- la progettazione hardware e software

dei sistemi per l'elaborazione delle informazioni basati
microprocessore (o CPU – Central Processing Unit)



Orario lezioni

Orario ufficiale:

<https://corsi.unibo.it/laurea/IngegneriaInformatica/orario-lezioni?anno=2&curricula=>

Lunedì 11.30-13.30, inizio ore 11.30 online

Giovedì 14-17, inizio ore 14.00 Aula 6.1

Materiale didattico

Lucidi in formato PDF:

<https://virtuale.unibo.it>

Prove d'esame e annunci (sito web del corso):

<http://vision.disi.unibo.it/~smatt/Site/Courses.html>

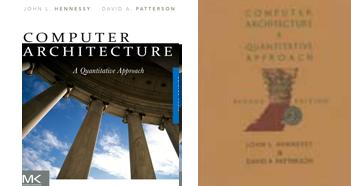
I lucidi non sono un libro, seguire con attenzione le lezioni è fondamentale per superare rapidamente e con buoni risultati l'esame...

Materiale didattico per approfondimenti

Seguendo il corso con attenzione e utilizzando i lucidi forniti **NON è necessario utilizzare altro materiale** per la preparazione dell'esame.

Tuttavia, per chi desiderasse approfondire:

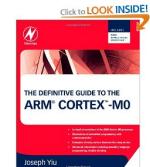
- Hennessy & Patterson, "Computer architecture: a quantitative approach", Morgan Kaufmann, Anche in versione italiana. La seconda edizione (inglese, a dx) descrive approfonditamente il processore DLX



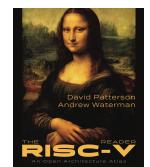
- G. Bucci, "Architettura e organizzazione dei calcolatori elettronici. Fondamenti", McGraw-Hill



- J. Yiu, "The definitive guide to the ARM Cortex M0", Newnes



- Patterson & Waterman, "RISC-V", Strawberry Canyon



Requisiti

Per superare in modo proficuo il corso di Calcolatori Elettronici T è fondamentale avere compreso bene:

- 1) **Fondamenti di Informatica T**
- 2) **Reti Logiche T**

Per Reti Logiche T è cruciale la progettazione *diretta*.

Si sconsiglia vivamente di seguire questo corso senza avere solide basi in 1) e soprattutto di **Reti Logiche T**

Avvisi e altre comunicazioni

Eventuali comunicazione di carattere generale saranno inserite nella pagina web del corso, nella sezione **"Avvisi"** di Calcolatori Elettronici T, chat Teams o annunci in Virtuale.

Calcolatori Elettronici T

Docente: Stefano Mattoccia (email: stefano.mattoccia@unibo.it)

Tutor: Matteo Poggi (email: matteo.poggi8@unibo.it)

Obiettivi: apprendere la struttura e il funzionamento dei calcolatori elettronici e sviluppare la capacità di progettare sistemi basati su microprocessore

Ricevimento studenti: su appuntamento concordato via email

Materiale didattico: disponibile su questa pagina web

Modalità d'esame: una prova scritta della durata di 2.5 ore. Durante l'esame non è possibile utilizzare dispositivi programmabili o dotati di qualsiasi capacità di comunicare (PC, tablet, smartphone, telefoni, calcolatrici programmabili, smartwatch, etc) e/o scattare foto. Inoltre, non è possibile consultare lucidi, appunti, libri o qualsiasi altro materiale didattico - in nessun formato - durante lo svolgimento dell'esame. Per sostenere l'esame è necessario essere in regola con l'iscrizione universitaria (verificata in Almaesami una volta chiusa la lista), avere il badge universitario e un documento di identità con fotografia e in corso di validità. Nella sezione FAQ, riportata in calce, le risposte alle domande più frequenti.

Avvisi

19/07/2018

La visione degli elaborati del 12/07/2018 e la verbalizzazione dei voti si svolgerà lunedì 23/07/2018 alle ore 13 in aula 5.7

18/07/2018

L'esito dell'esame del 12/07/18 è disponibile in Almaesami.

La data per la visione degli elaborati e la verbalizzazione sarà indicata nei prossimi giorni.

Modalità di svolgimento dell'esame

- L'esame consiste in una prova **SCRITTA di 2.5 ore**
- **Nessuna prova orale**
- Non è possibile portare libri, appunti, computer, telefoni, smartphone, tablet, smart watch, etc
- E' indispensabile (pena l'esclusione dall'esame) presentarsi con **documento di identità e badge**
- **Esami gravemente insufficienti saranno verbalizzati**

Prossimi appelli d'esame

Se possibile (aula, etc) il primo appello subito dopo il termine delle lezioni

- La date degli esami sono consultabili su Almaesami
- L'**iscrizione** agli appelli via Almaesami è obbligatoria e si chiude (circa) una settimana prima
- Non è ammessa alcuna deroghe all'iscrizione
- Sono previsti 6 appelli all'anno:
 - 3 Dicembre/Febbraio
 - 2 Giugno/Luglio
 - 1 Settembre

Nessun appello straordinario

Struttura di una prova d'esame

Esame di Calcolatori Elettronici T 22 Dicembre 2017 (Ing. Informatica)

Esercizio 1

In un sistema basato sul DLX, con 768 MB di EPROM mappata negli indirizzi bassi e 128 MB di RAM mappata negli indirizzi alti, sono presenti: una porta a 8 bit in input (IN_1) e quattro porte a 8 bit in output (denominate OUT_3, OUT_2, OUT_1 e OUT_0). Tutte le porte in input e output utilizzano il protocollo di handshake.

Dopo ogni trasferimento dalla porta in input deve essere eseguito, appena possibile, un unico trasferimento sulla porta in output OUT_i prima di poter eseguire una successiva lettura dalla porta in input. L'indice i, che determina la porta in output sul quale eseguire il trasferimento successivo alla lettura dalla porta in input, corrisponde al resto della divisione per 4 del dato precedentemente letto (dalla porta in input). Tutte le altre porte in output non debbono essere utilizzate finché non risulteranno abilitate mediante la strategia appena delineata (i.e., lettura porta in input -> scrittura sull'unica porta in output abilitata -> lettura da porta in input -> scrittura sull'unica porta in output abilitata -> etc). Il dato letto dalla porta in input deve essere scritto in memoria a FFFF0080h mentre il dato scritto sulla porta in output abilitata deve essere letto a FFFF0100h. All'avvio del sistema deve essere abilitato come primo trasferimento quello che coinvolge la porta in input.

- Descrivere sinteticamente la soluzione che s'intende realizzare e indicare chiaramente quali sono i segnali di chip-select necessari
- Progettare il sistema, minimizzando le risorse necessarie evidenziando e risolvendo eventuali criticità
- Indicare le espressioni di decodifica e il range di indirizzi di tutte le periferiche, le memorie e i segnali
- Scrivere il codice ottimizzato dell'interrupt handler (i registri da R25 a R29 possono essere utilizzati senza la necessità di doverli ripristinare)
- Soluzioni interamente software NON saranno considerate valide

Esercizio 2

Che cosa sono gli interrupt annidati? Quali problematiche comporterebbero nel caso del DLX?

Esercizio 3

Che cosa s'intende per architettura Harvard? Quali problemi mira a risolvere e come può essere realizzata nel caso del DLX?

Esercizio 1: progetto di un sistema a microprocessore.

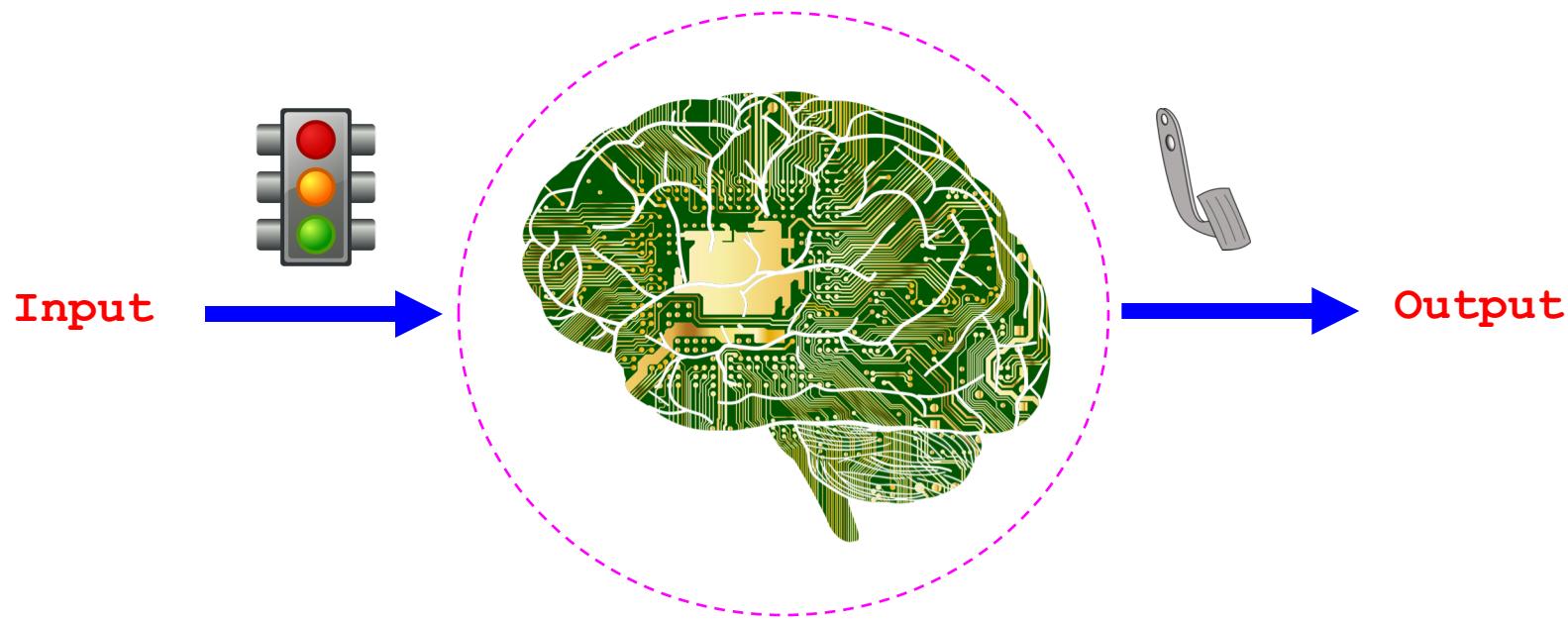
Per superare l'esame, è necessario proporre una soluzione ragionevole.

L'esercizio 1 determina il superamento/non superamento della prova.

Esercizi 2 e 3: domande di teoria che richiedono qualche ragionamento. Volutamente non è fornita la soluzione.

Elaborazione delle informazioni

Un essere vivente elabora continuamente delle informazioni e fornisce delle risposte o agisce in un determinato modo in base alle informazioni in ingresso.



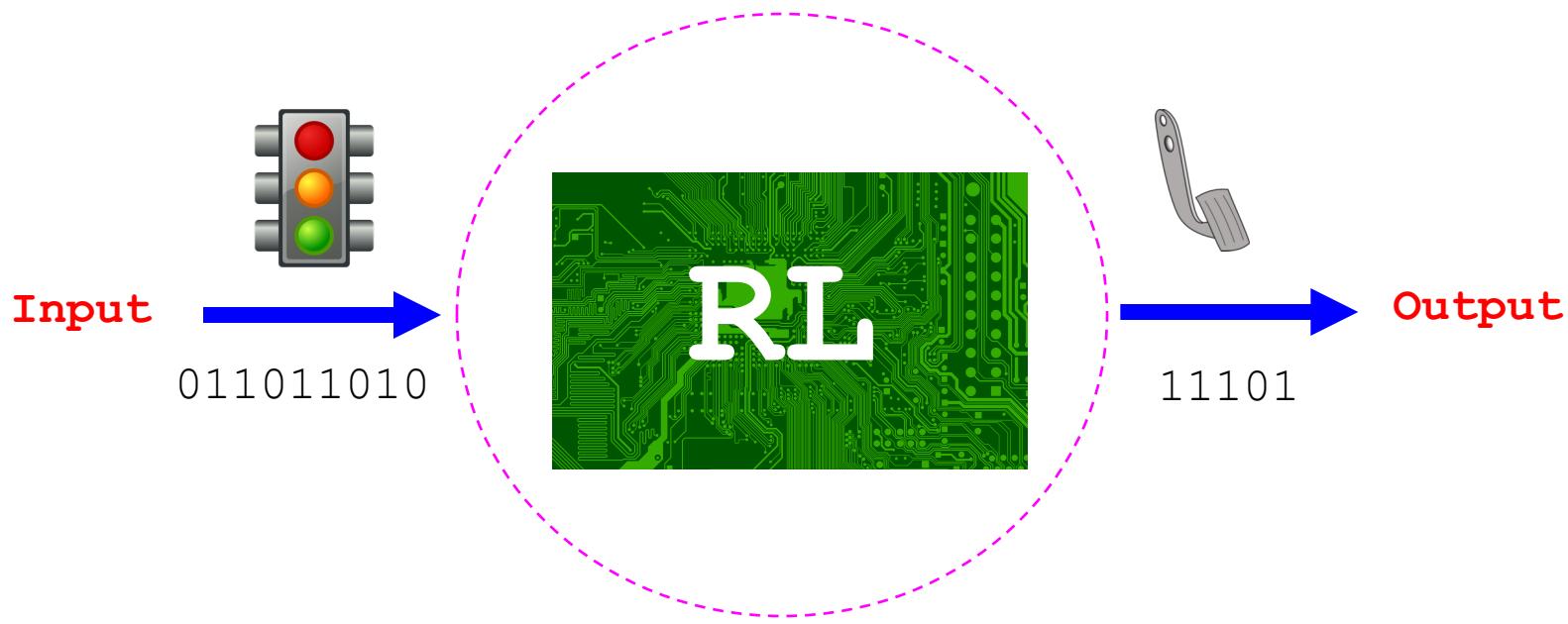
Un sistema di qualsiasi natura per l'elaborazione delle informazioni isolato dall'esterno servirebbe a ben poco (meglio, nulla).

Input e output debbono essere codificati in modo appropriato

Elaborazione delle informazioni

Ovviamente, in questo corso, siamo interessati a sistemi di natura elettronica per elaborare le informazioni.

In particolare, per le ragioni evidenziate a Reti Logiche T, siamo interessati a **sistemi digitali** -> Reti Logiche



Elaborazione delle informazioni

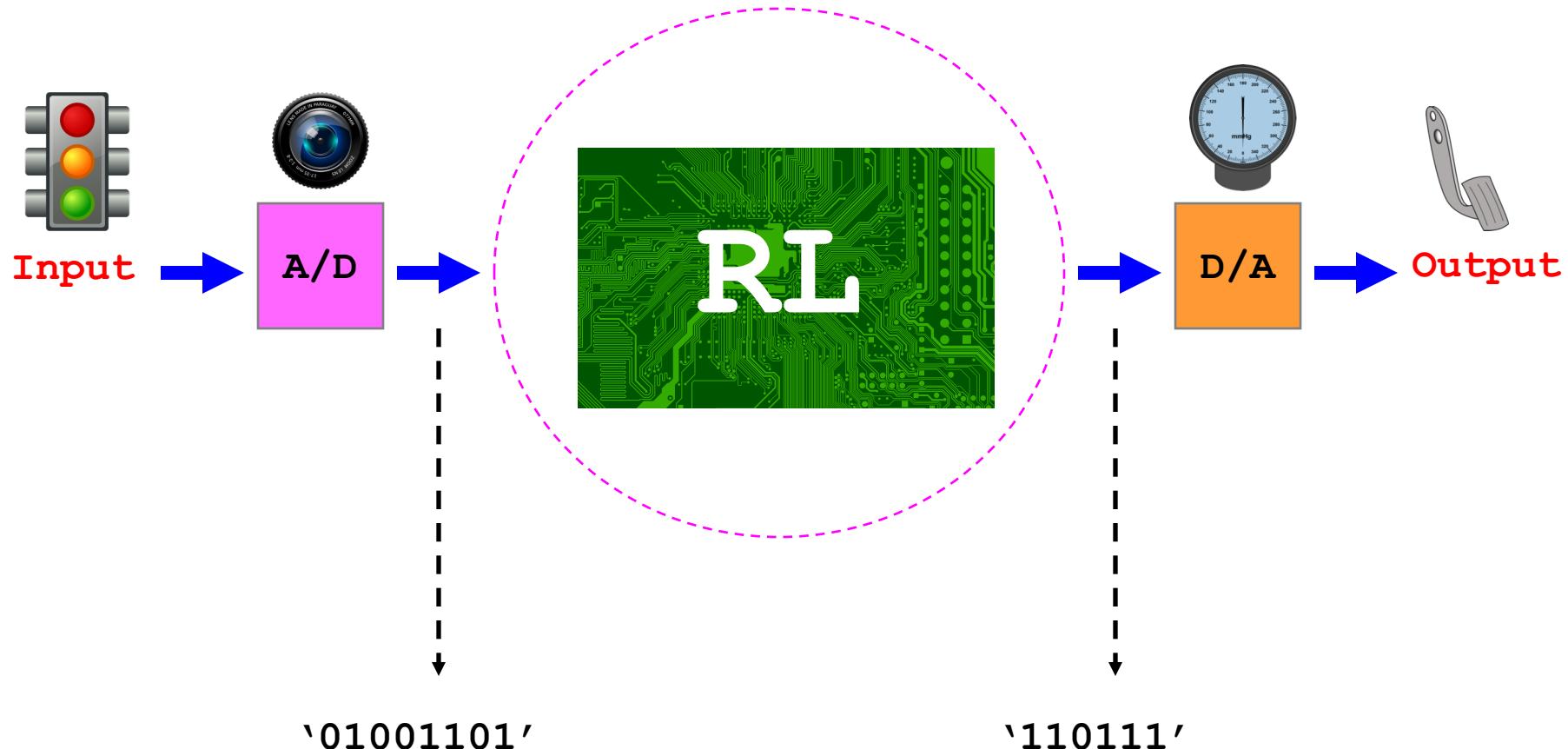
Un primo problema: le reti logiche elaborano informazioni di tipo digitale

Molti input e output di particolare interesse non sono di natura digitale ma analogica (quella preferita dagli esseri umani)

- Prossimità a un semaforo o un ostacolo
- Monitor
- Movimento del mouse
- Voce/Audio
- Pressione tasti
- Touch screen
- etc

Elaborazione delle informazioni

Pertanto, è (spesso) necessario convertire segnali di natura analogica in segnali digitali e viceversa



In un sistema di elaborazione, in ingresso e in uscita troviamo solo segnali digitali (binari)

Elaborazione delle informazioni

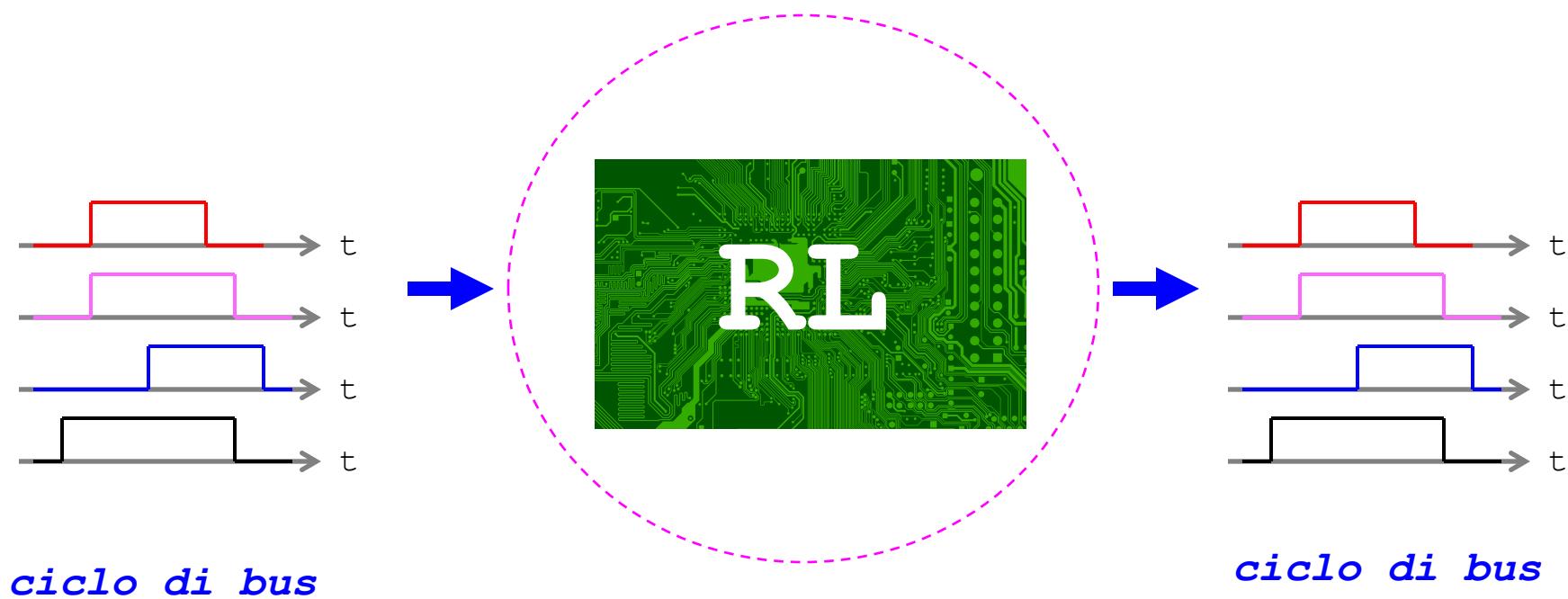
Altro problema: la rete logica che elabora le informazioni è spesso molto efficienti/veloce/etc ma allo stesso tempo poco flessibile per quanto riguarda il *linguaggio utilizzabile* per fornire le informazioni (digitali) in input e *interpretare* le informazioni (digitali) elaborate in output.

Se si desidera interagire con un sistema di elaborazione digitale è necessario adeguarsi al *linguaggio* che la rete logica utilizza.

Questo linguaggio è codificato dall'evoluzione temporale di alcuni segnali (il minimo indispensabile) emessi o ricevibili dalla rete logica. L'evoluzione temporale di questi segnali si definisce *ciclo di bus (bus cycle)*.

Elaborazione delle informazioni

Quindi, l'unico modo per interagire con un sistema digitale per l'elaborazione delle informazioni consiste nell'adottare questa convenzione (ie, i cicli di bus, descritti in dettaglio sui datasheet che il produttore del sistema rende sempre disponibili).



Quale rete logica utilizzare?

Sono disponibili diverse architetture di elaborazione (i.e., reti logiche). Quale scegliere?

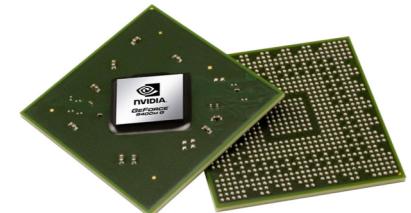
Dipende dal contesto applicativo e dalle prestazioni, consumi, ingombri, peso, etc:

- sistema *general purpose*
- sistema *embedded*
- sistema per videogiochi
- etc

Una tipologia di architettura, basata sul **modello di Von Neumann**, è molto più flessibile delle altre e può essere utilizzata, anche se non sempre con risultati ottimali, in ogni contesto. Questo modello è l'oggetto di questo corso e utilizza come elemento di base una CPU (**microprocessore**).

Esempi di architetture di elaborazione

- CPU (Central Processing Unit)
- CPU Multicore
- CPU Embedded
- SOC (System on a Chip)
- GPU (Graphic Processing Unit)
- FPGA (Field Programmable Gate Array)
- Sistemi ibridi (e.g. FPGA + CPU)
- . . .

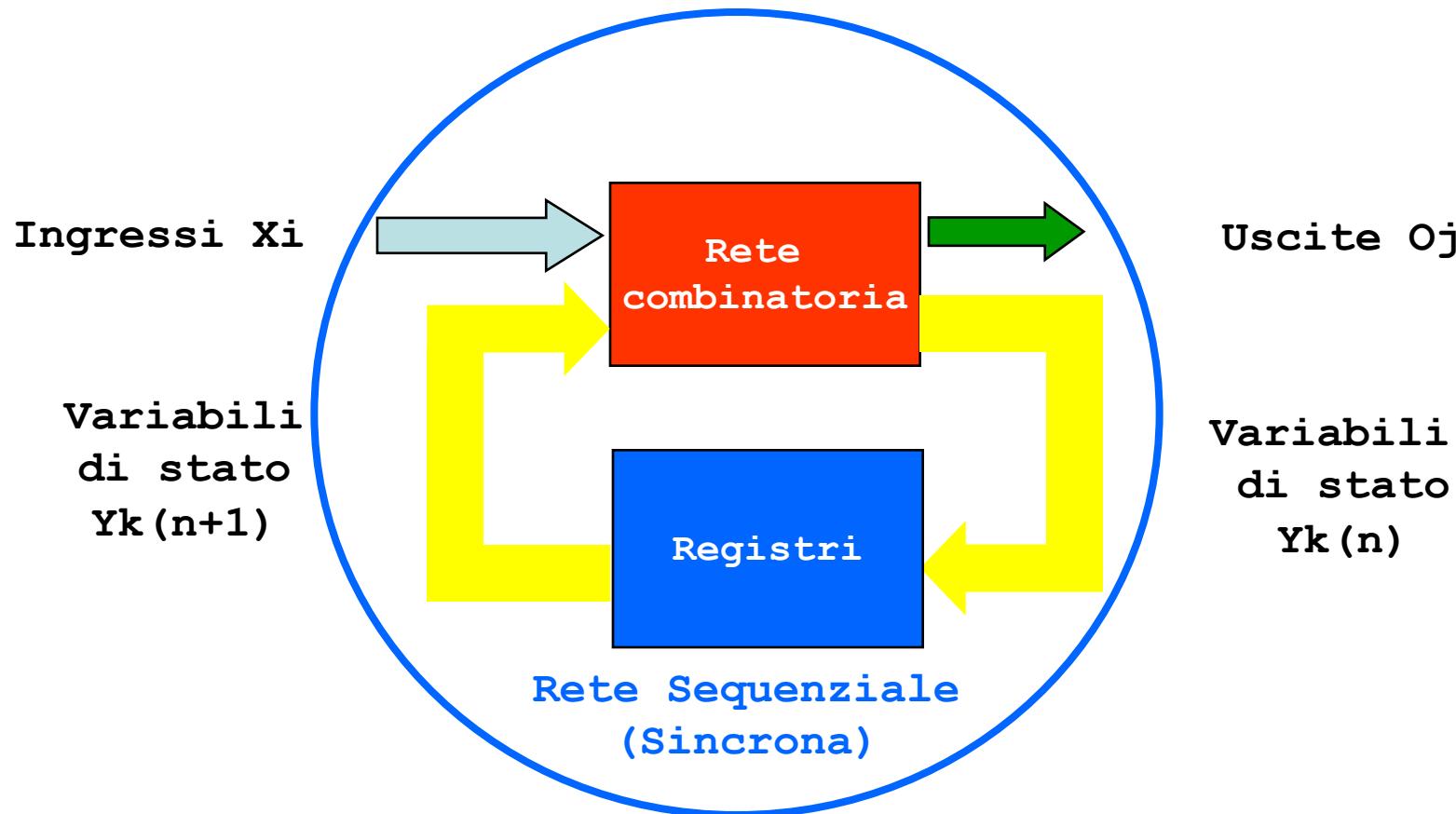


Il modello di riferimento: Von Neumann

- Architettura del tutto generale che porta a realizzazioni poco dipendenti dal funzionamento desiderato
- Il funzionamento desiderato è espresso in termini di
 - * sequenza di istruzioni (programma)
 - * memorizzate su un supporto di memoria
- Per cambiare funzionamento è sufficiente cambiare il programma: questo di fatto “modifica” la rete logica di controllo per ogni istruzione eseguita
- L'architettura è adatta a trattare problemi molto più complessi di quelli visti nel corso di reti logiche ma con efficienza (tipicamente) inferiore
- L'importanza e la diffusione dei calcolatori dipende fortemente dalla *flessibilità* di questo modello

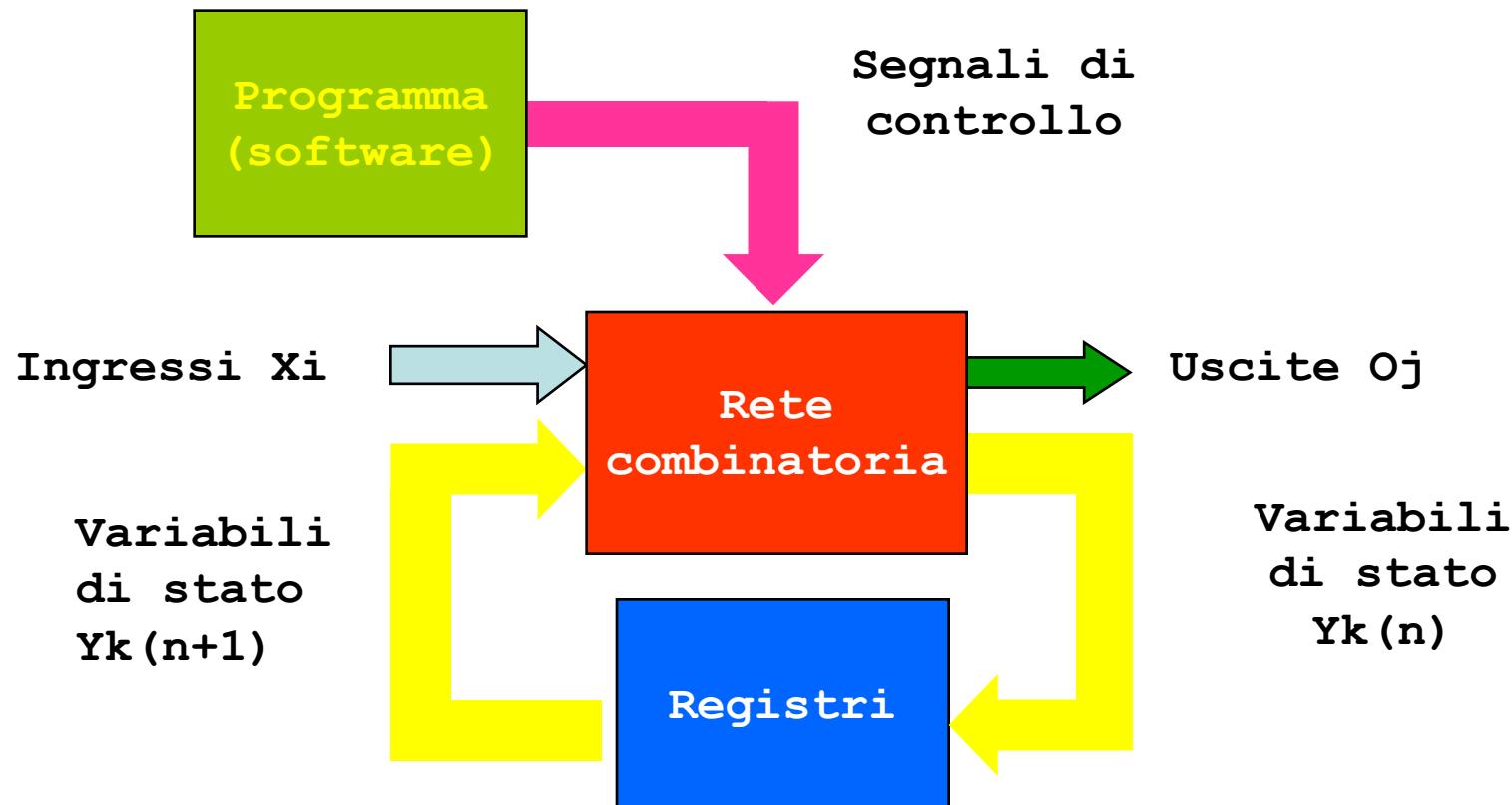
Sistema di elaborazione: rete sequenziale

Il sistema di elaborazione può essere schematizzato in modo astratto come una **Rete Sequenziale (Sincrona)**, RSS

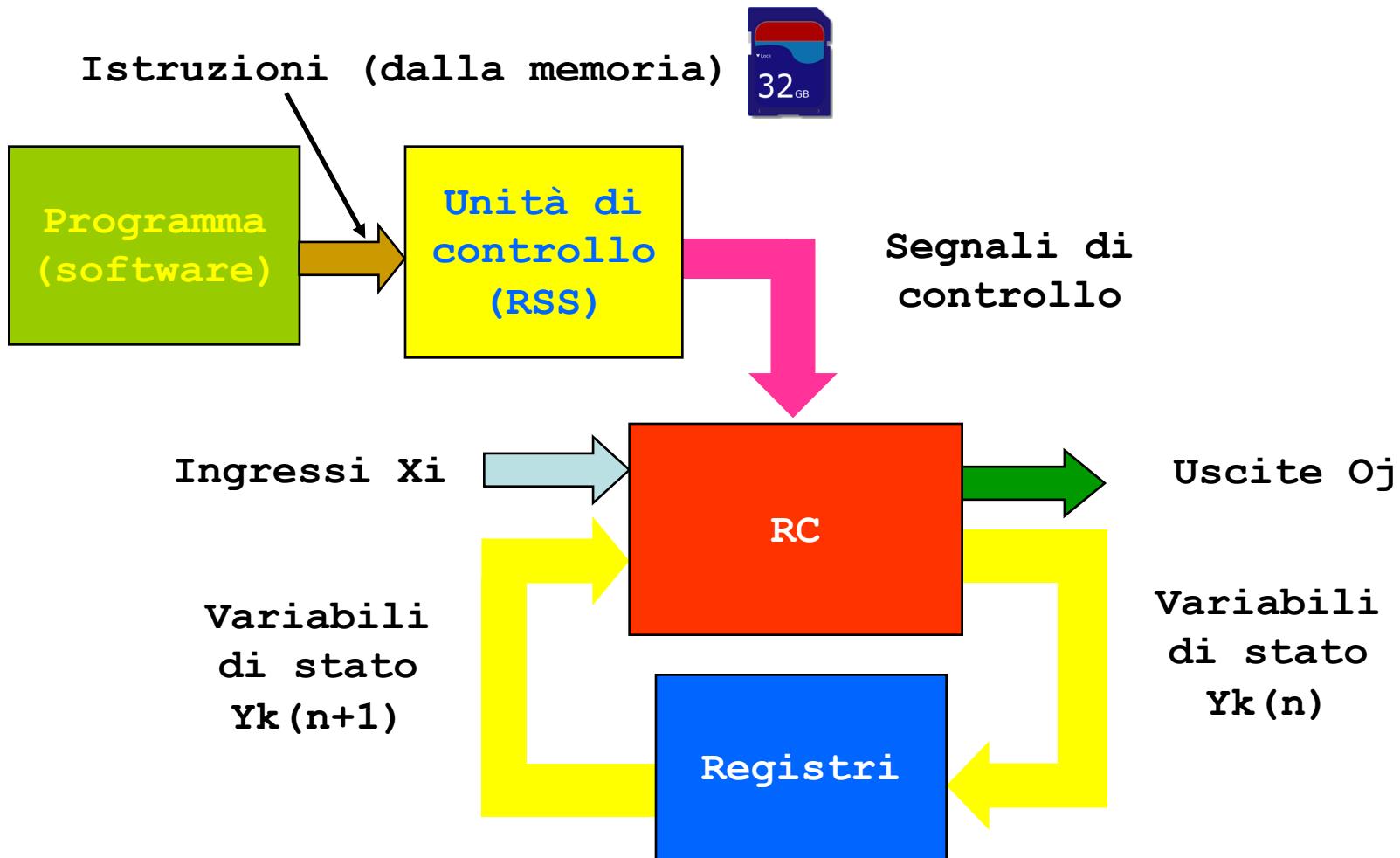


Come cambiare il funzionamento della RL?

Rispetto a Reti Logiche, la novità è il **software** (programma) che consente di variare il funzionamento della rete in base alle esigenze desiderate (ie, il codice scritto dal programmatore).



In realtà le cose sono più complesse

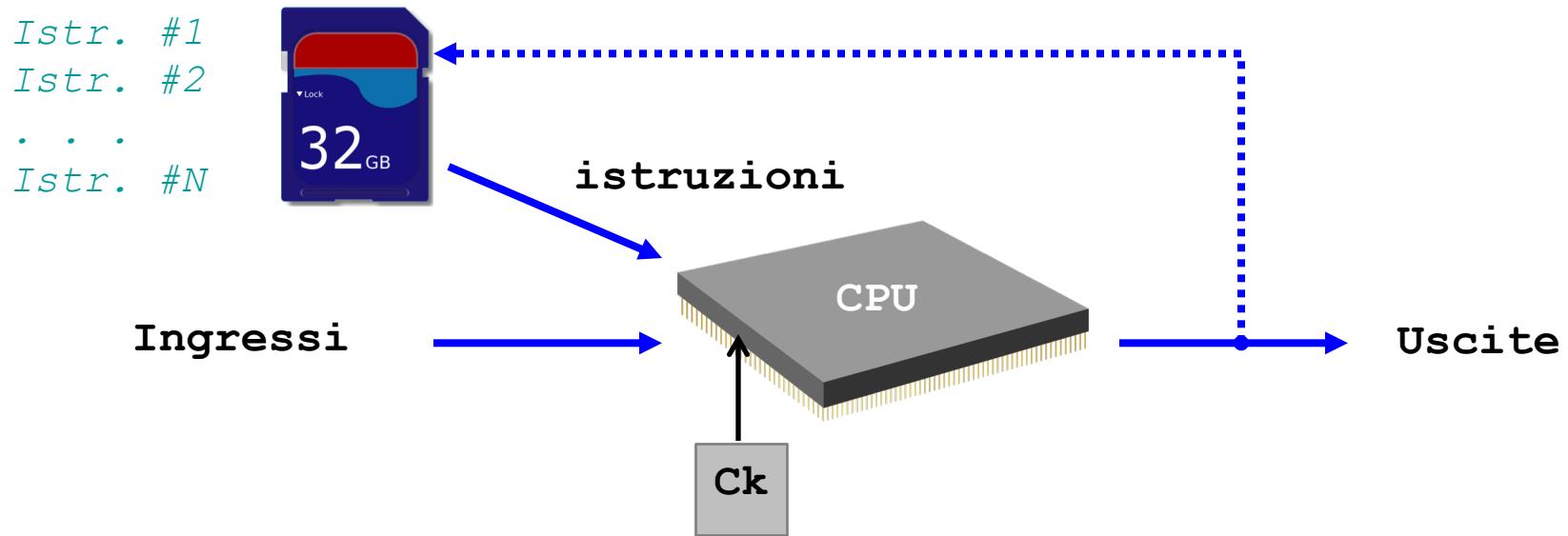


L'unità di controllo non solo governa la rete combinatoria ma anche tutte le altre reti logiche presenti nel sistema

Esempio: abilita gli ingressi e le uscite quando necessario, etc

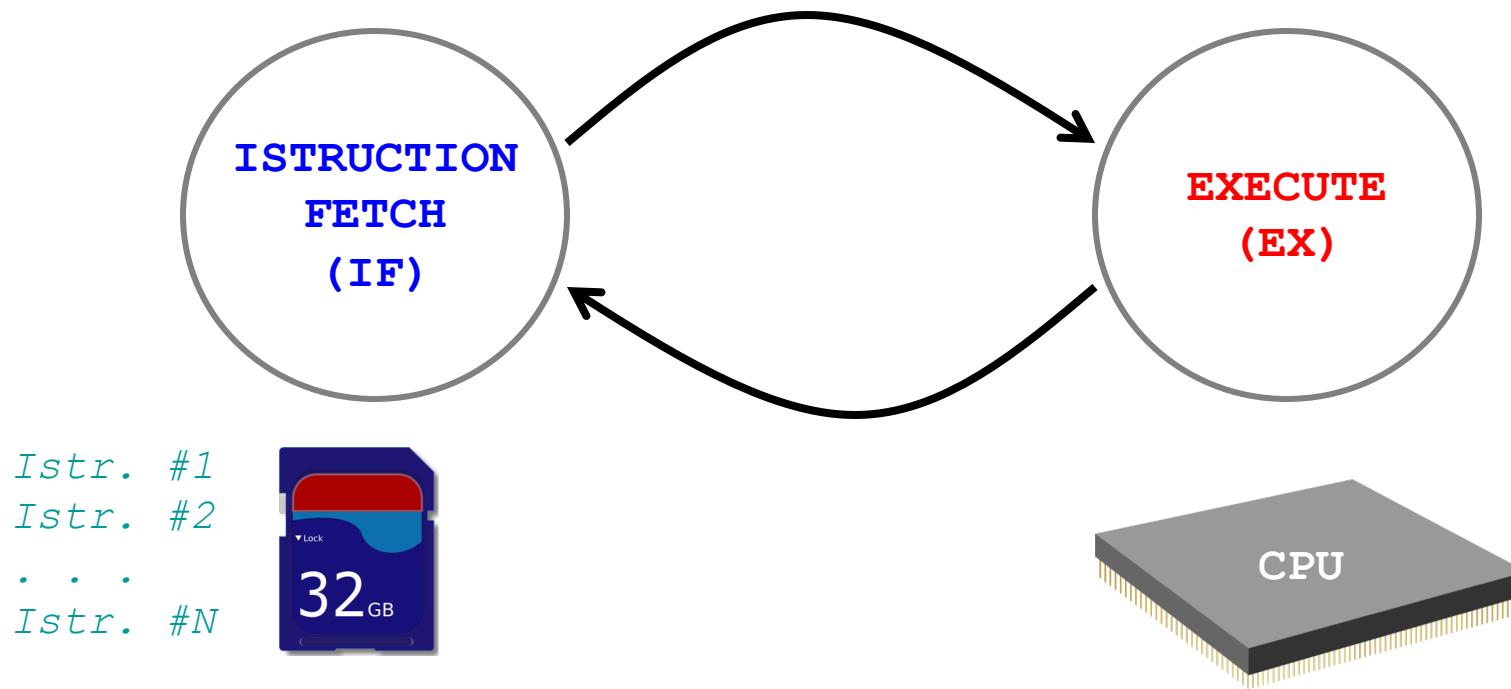
Modello di esecuzione del programma

- Il programma risiede in memoria ed è costituito da istruzioni codificate in forma binaria (linguaggio macchina)
- In memoria risiedono anche gli operandi delle istruzioni, cioè i dati elaborati e da elaborare (forma binaria)
- Le istruzioni vengono eseguite in sequenza dalla CPU
- La CPU è una macchina sequenziale sincrona (con un clock)



A livello di massima astrazione, il funzionamento dell'intero sistema può essere descritto mediante due soli stati:

- Stato in cui la CPU legge in memoria la prossima istruzione da eseguire (**INSTRUCTION FETCH** o IF)
- Stato in cui la CPU esegue l'istruzione letta in IF (**EXECUTE** o EX)



RL CPU e istruzioni

- Le reti logiche viste a reti logiche elaboravano e producevano segnali binari
- Se la CPU è una RL, come sono codificate le istruzioni?
- Ovviamente in binario...
 - Esempio

istruzione #1 → 00010100000101111101010000010011

istruzione #2 → 10110101100100011001010000011001

... . . .

... . . .

istruzione #N → 01010110100101010101010110011110

- Ogni istruzione indica alla CPU quale operazione deve svolgere/eseguire
- Non sembra essere un modo molto comodo (per gli esseri umani)

Esempi di istruzioni

- Quali istruzioni possono essere eseguite da una CPU?
 - Somme
 - Moltiplicazioni
 - Divisioni
 - Confronto
 - Letture dalla memoria o da altri dispositivi*
 - Scritture in memoria o verso altri dispositivi*
 - . . .

Esistono sostanzialmente due tipologie di CPU:

- **RISC** (Reduced Instruction Set Computer)
Poche e semplici istruzioni, reti logiche semplici e molto veloci (frequenza di clock elevata). eg, ARM
- **CISC** (Complex Instruction Set Computer)
Molte istruzioni, alcune molto complesse, reti logiche complicate. eg, Intel e AMD

RISC vs CISC

- Tipicamente a una sola istruzione CISC corrispondono più istruzioni RISC
- Tuttavia, ogni singola istruzione RISC è eseguita spesso più rapidamente di una istruzione CISC
- Spesso, il codice RISC, anche se più denso, è più veloce
- Le RL RISC sono tipicamente più semplici di quelle CISC
- Se le reti sono più semplici lo spazio (silicio) può essere utilizzato per altre finalità (registri, cache, etc)
- I processori RISC sono molto diffusi (smartphone, tablet)
- Anche i processori CISC sono molto diffusi (PC) per via del software esistente*
- Le CPU CISC moderne sono in realtà internamente dei RISC

Istruzioni in forma più comprensibile

- Indipendentemente dal tipo di CPU, le **istruzioni in forma binaria** non sono **facilmente interpretabili** e per questo non utilizzate in questa forma quando si scrive il codice
- Il linguaggio che si utilizza è **l'assembler**:

ADD R1,R2,R3; pone in R1 la somma tra registri R2 e R3

Questa istruzione potrebbe essere codificata con:

00010100000101111101010000010011

Il traduttore assembler -> codice macchina in binario è una Look Up Table (LUT), ovvero una tabella

L'assembler sembra essere un passo avanti notevole ma...

Perché non avete mai utilizzato il linguaggio assembler nonostante scrivete codice dal primo anno?

Compilatore e istruzioni binarie

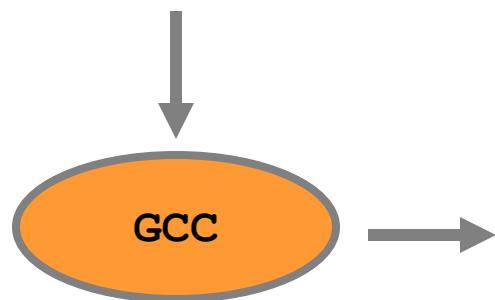
- Il motivo è che avete scritto codice ad alto livello (C) e utilizzato un compilatore (e.g., GCC)
- Il compilatore, converte il codice scritto in linguaggio ad alto livello in istruzioni macchina binarie

```
#include <stdio.h>
int main(int argc, char **argv)
{
    int a=5;
    int b=6;
    int c;

    c = a + b;

    printf("La somma tra %d e %d è %d\n", a, b, c);
}
```

SPESO NEI SISTEMI EMBEDDED IL CODICE VIENE PRIMA COMPILATO SU PC, CHE GENERA LINGUAGGIO MACCHINA PER IL DISPOSITIVO DI DESTINAZIONE



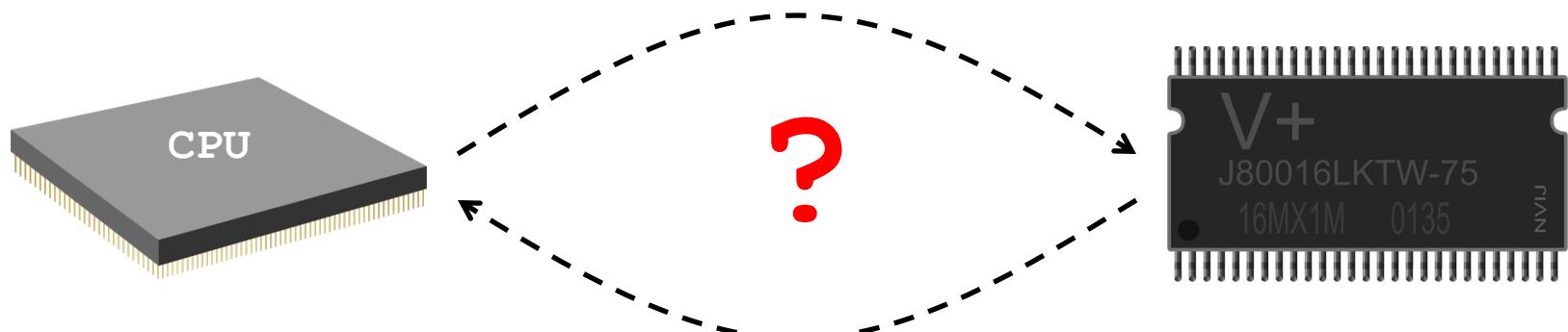
```
00010100000101111101010000010011
10110101100100011001010000011001
...
01010110100101010101010110011110
```

Come si accede alla memoria (e non solo) ?

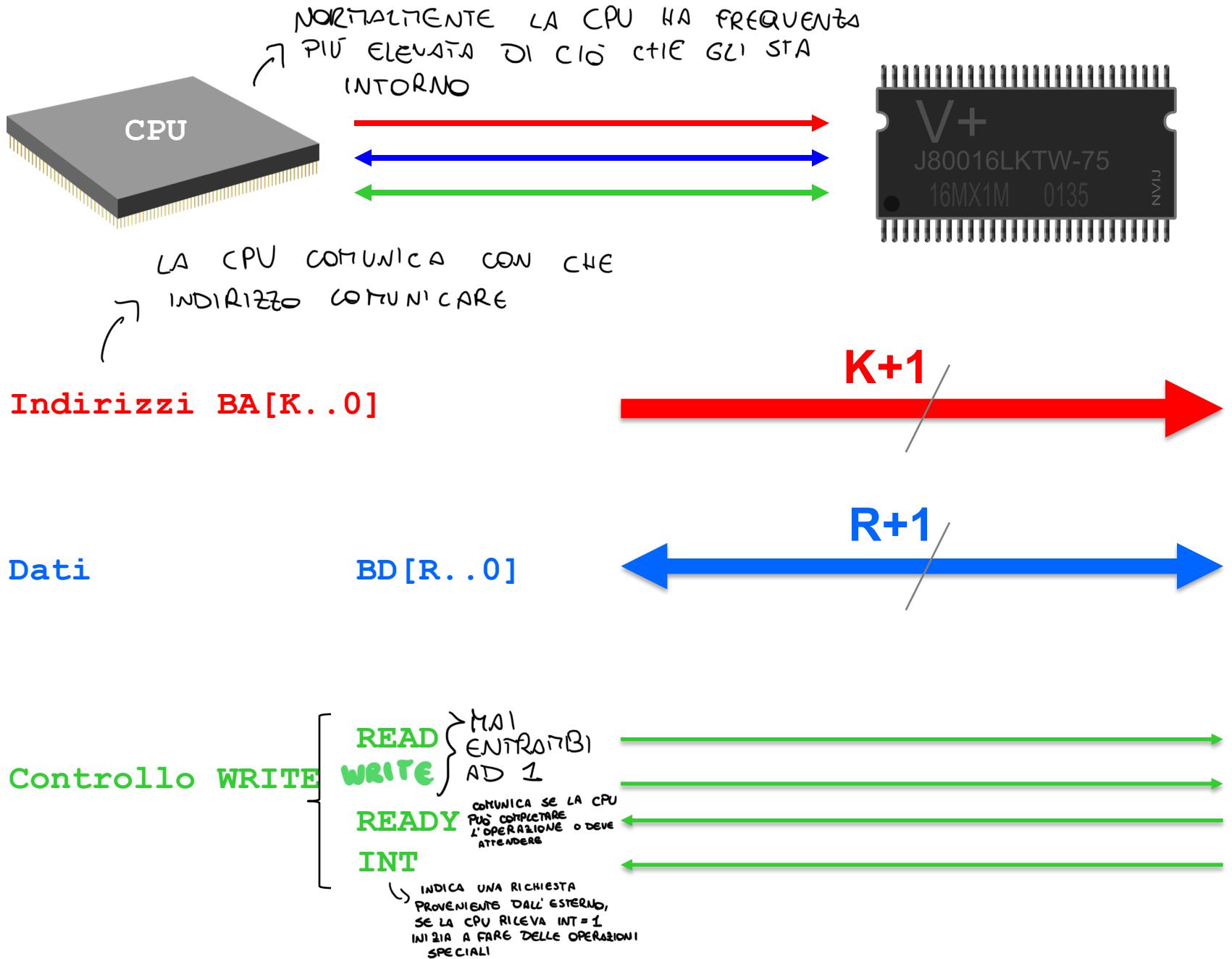
- Sappiamo che il codice nel modello di Von Neumann risiede in memoria
- Come si legge (e scrive dalla memoria) ?
- Come si leggono e scrivono i dati?

CIASCUA ISTRUZIONE DEVE ESSERE LETTA DALLA MEMORIA E PORTATA NELLA CPU, CHE POI DECODIFICA LE ISTRUZIONI

ED POI I DATI VENGONO RIPORTATI NELLA MEMORIA



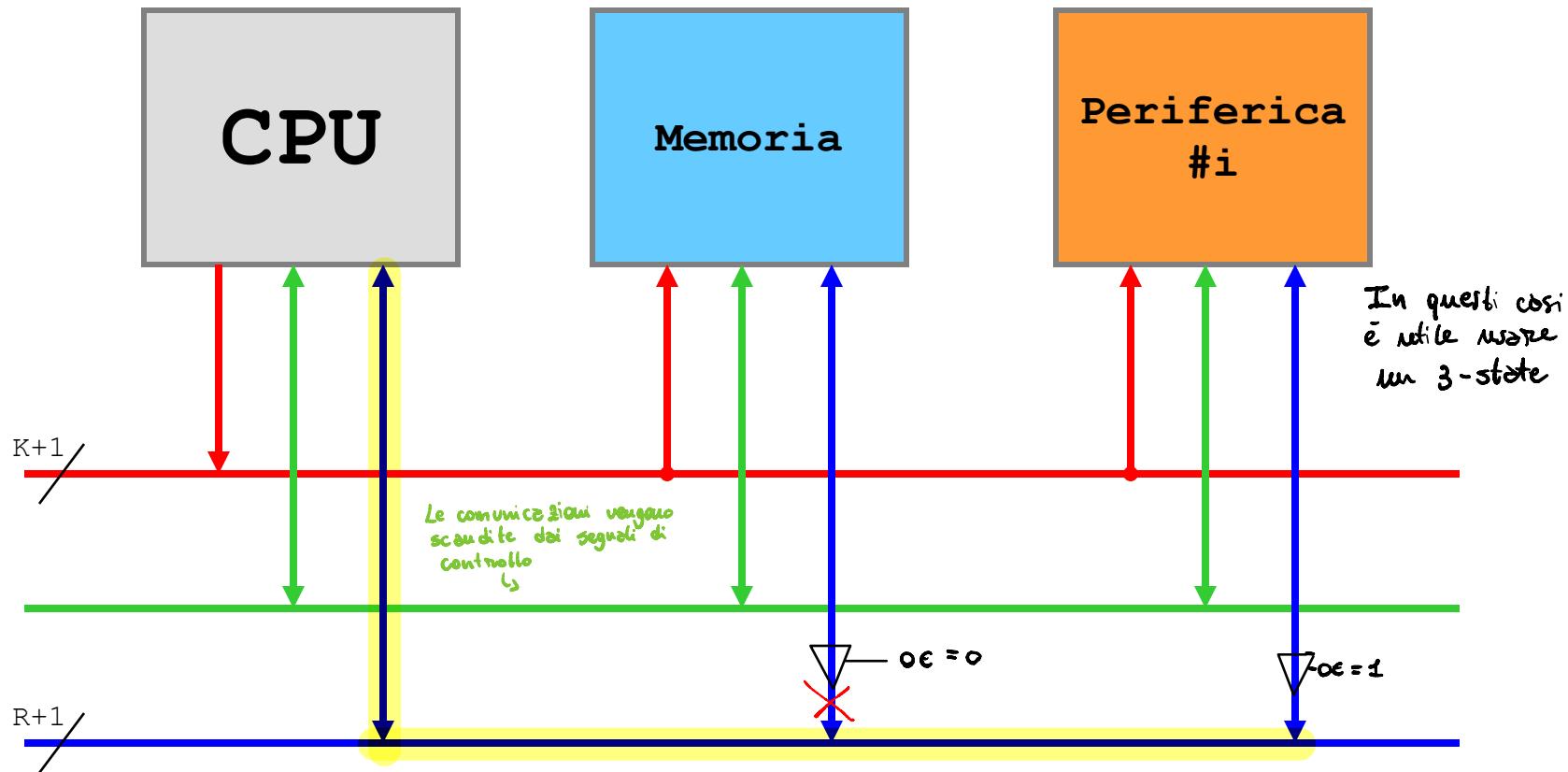
- L'unico modo è attraverso dei segnali predefiniti (con un ben definito andamento temporale, **ciclo di bus**)



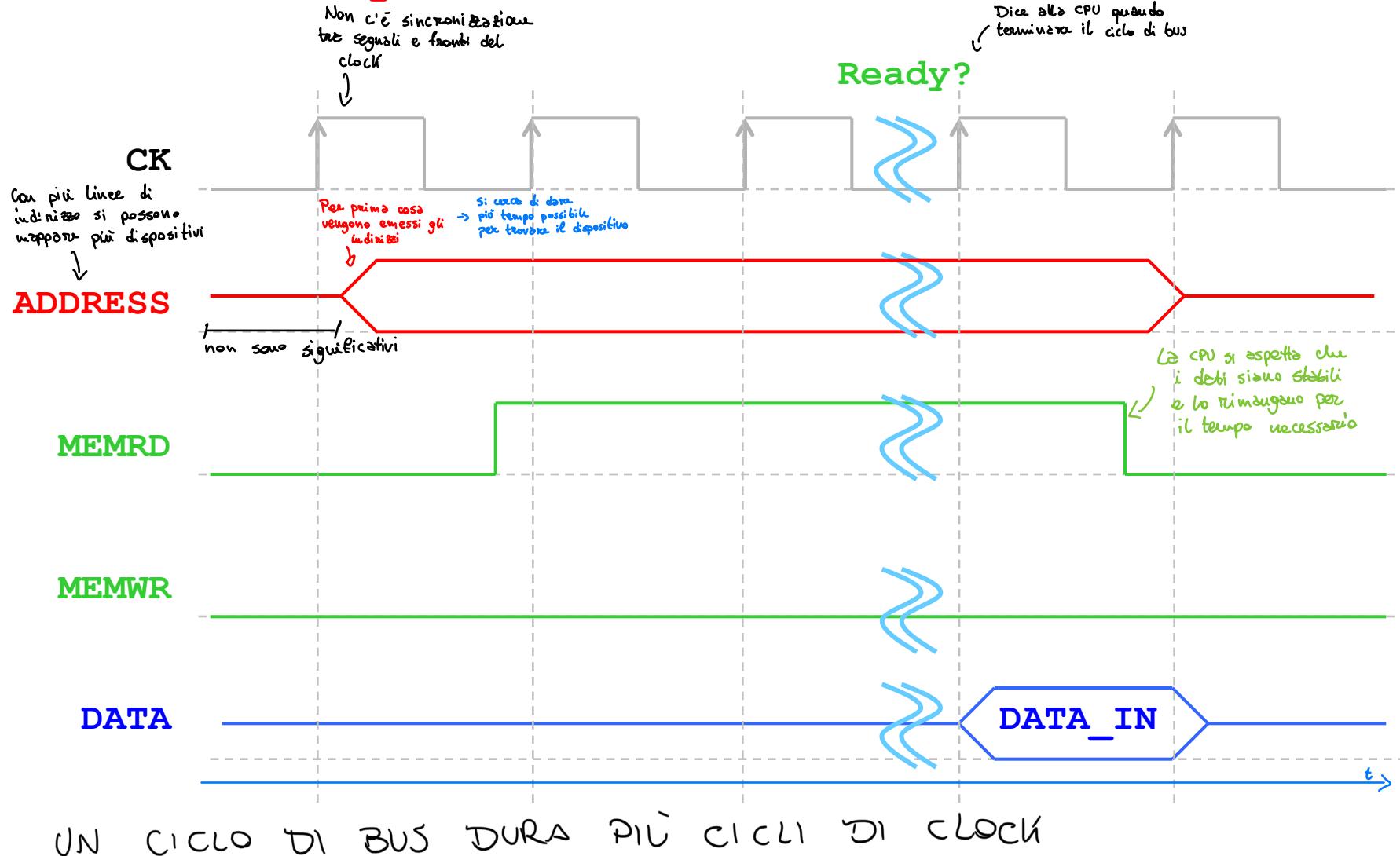
Come avviene la comunicazione?

- Tutto viaggia sui bus di sistema
- Tutto è regolato da cicli di bus

GLI INDIRIZZI VENGONO
ASSEGNAȚI QUANDO VIENE PROGETTATO
IL SISTEMA

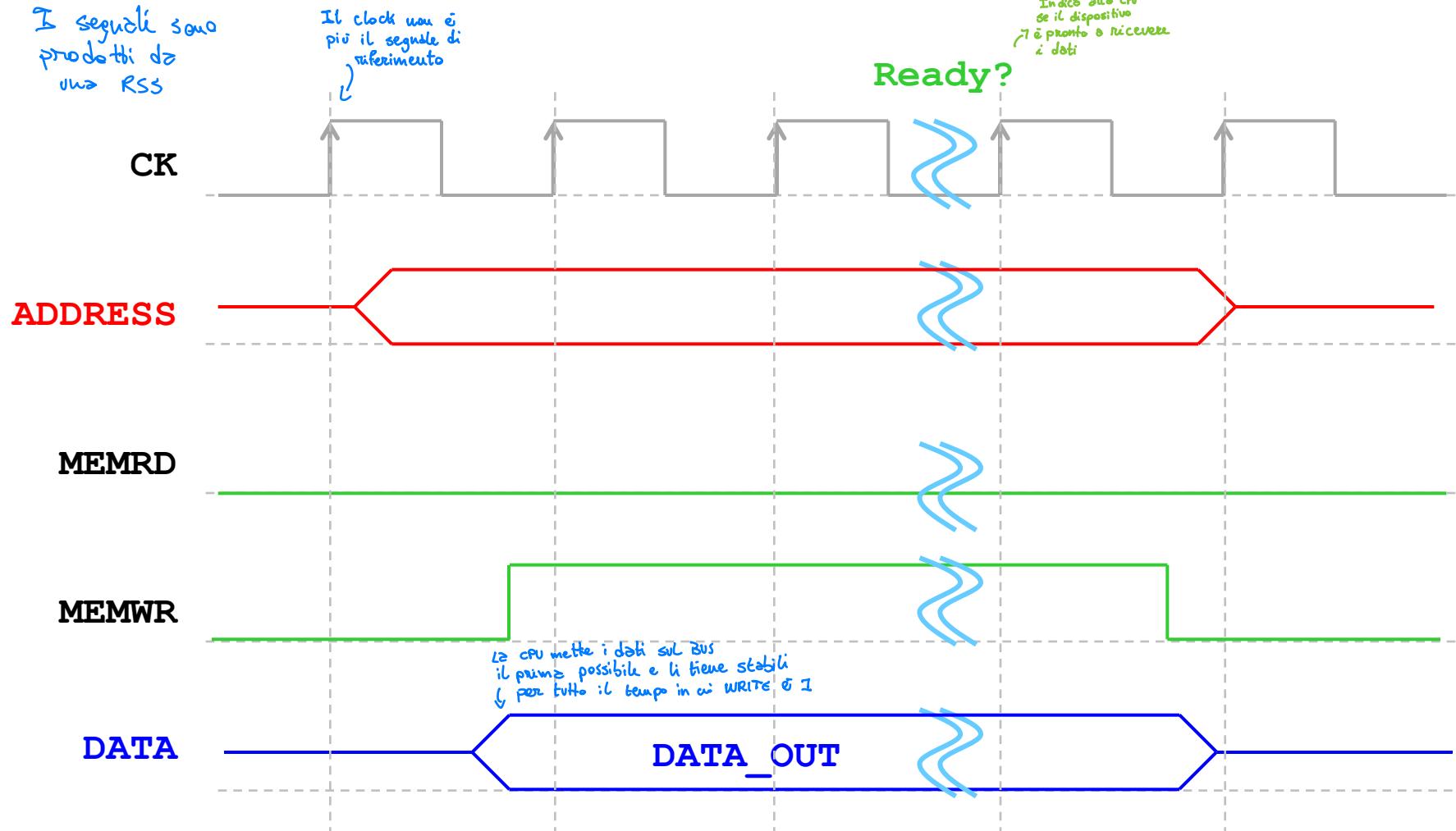


Esempio di ciclo di lettura



La CPU sta facendo un ciclo di bus o per fare un fetch o perché viene richiesto dal codice del software

Esempio di ciclo di scrittura



Meno la CPU comunica con l'esterno e meglio è perché lavora più velocemente

Range di indirizzi

- Se sono disponibili 32 bit (BA[31..0]) è possibile avere accesso a 2^{32} elementi (memorie, periferiche)
- 32 segnali di indirizzo, 4 GB
- A ogni indirizzo è associato un dispositivo (memorie, periferiche)
- E' necessario decodificare l'indirizzo emesso dalla CPU per determinare con quale dispositivo la CPU intende comunicare
- Quanti byte possono essere trasferiti durante un ciclo di bus? Dipende dall'ampiezza del bus dati

La quantità di dati che si possono trasferire dipende dal numero di fili disponibili

Relazione tra hardware e software

- Consideriamo una istruzione per leggere dalla memoria un byte a un determinato indirizzo

xxx LB destinazione, indirizzo ; lettura di un byte

Per prima cosa è eseguito il fetch dell'istruzione all'indirizzo **xxx**. come? Con un ciclo di bus, naturalmente

Come facciamo a conoscere l'indirizzo **xxx**?

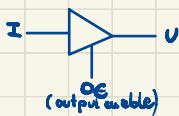
La CPU ha accesso al **program counter PC**

Una volta letta e decodificata, l'istruzione è eseguita.

Durante l'esecuzione è eseguito un ciclo di bus di lettura

Durante il ciclo di bus è emesso l'**indirizzo**

DRIVER 3-STATE

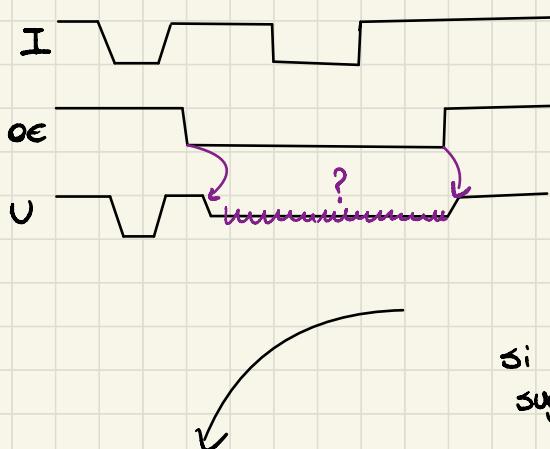


OE	I	U
1	0	0
1	1	1
0	0	2
0	1	2

Può essere usato per rigenerare un segnale, dal punto di vista logico non fa nulla

Il segnale OE fa comportare il dispositivo da buffer se $OE = 1$.

Se $OE = 0$ il dispositivo disconnecte dal punto di vista elettrico il segnale di ingresso dall'uscita

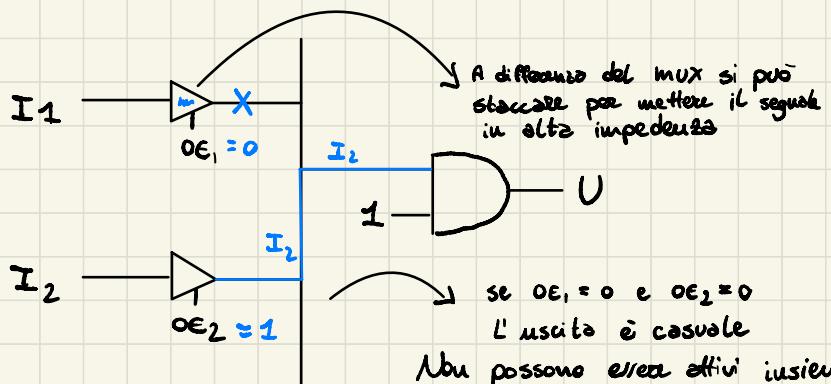
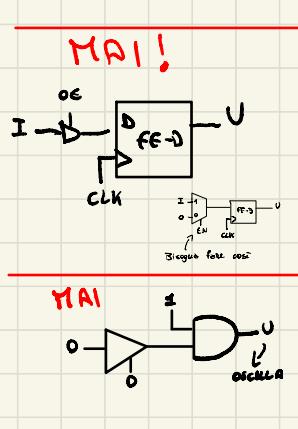


Se $OE = 0$ si dice che il segnale è in alta impedenza

Il 3-state è utile per disconnectare temporaneamente un segnale così da rendere significativo un altro.

Si possono connettere più dispositivi sugli stessi fili in momenti diversi

La CPU può trasferire dati su dispositivi diversi in momenti diversi



se $OE_1 = 0$ e $OE_2 = 0$
l'uscita è casuale
Non possono essere attivi insieme perché crearebbero un cortocircuito